The artifact I'm submitting are the changes based on Milestone 4's feedback and also the database items. The database items will include the actual MongoDB database with Jupyter Notebook as well (Jupyter was an easy way to be able to show you that the database was actually created – I did seed it via Python Django, however, there's no image that the database was created correctly. So Jupyter is just a way for your assurance).

With feedback given from the previous module, I first ended up updating **models.py** to incorporate some different elements. Under the Doctor class, you'll now see a "specialty". The "Patient" class received an update to include the assigned doctor/nurse to the patient

```python
class Patient(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    patient_id = models.AutoField(primary_key=True)
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    date_of_birth = models.DateField()
    contact_information = models.TextField()
    assigned_doctor = models.ForeignKey('Doctor', on_delete=models.SET_NULL, null=True)
    assigned_nurse = models.ForeignKey('Nurse', on_delete=models.SET_NULL, null=True)


class Doctor(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    doctor_id = models.AutoField(primary_key=True)
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    specialty = models.CharField(max_length=100)
    contact_information = models.TextField()

    def __str__(self):
        return f'Dr. {self.first_name} {self.last_name}'
```

Also, everyone else received minor updates as well. The MedicalRecord class I tagged in the doctor, nurse, diagnosis, treatment, and date_of_record. Now looking back on this, though I commented out for this task, It's best that I incorporate back in the note as logically, it doesn't make sense to leave it out of the entire record, but the whole record doesn't need to show within the note.

```python
class MedicalRecord(models.Model):
    record_id = models.AutoField(primary_key=True)
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE)
    #notes = models.ManyToManyField('MedicalNote', blank=True)
    #created_at = models.DateTimeField(auto_now_add=True)
    #updated_at = models.DateTimeField(auto_now=True)
    doctor = models.ForeignKey(Doctor, on_delete=models.SET_NULL, null=True)
    nurse = models.ForeignKey(Nurse, on_delete=models.SET_NULL, null=True)
    diagnosis = models.TextField()
    treatment = models.TextField()
    date_of_record = models.DateField()

    def __str__(self):
        return f'Record {self.record_id} for {self.patient}'

class MedicalNote(models.Model):
    note_id = models.AutoField(primary_key=True)
    medical_record = models.ForeignKey(MedicalRecord, on_delete=models.CASCADE)
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    note_content = models.TextField()
    date_of_note = models.DateField(auto_now_add=True)

    def __str__(self):
        return f'Note {self.note_id} by {self.author} on {self.date_of_note}'

class Appointment(models.Model):
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE, null=True, blank=True)
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE)
    #date = models.DateField()
    #time = models.TimeField()
    #status = models.CharField(max_length=20, choices=(('requested', 'Requested'), ('confirmed', 'Confirmed'), ('declined', 'Declined')))
    appointment_id = models.AutoField(primary_key=True)
    appointment_date = models.DateTimeField()
    reason = models.TextField()
    is_booked = models.BooleanField(default=False)

    def __str__(self):
        return f'Appointment {self.appointment_id} with Dr. {self.doctor.last_name} for {self.patient.last_name if self.patient else "N/A"} on {self.appointment_date}'
```

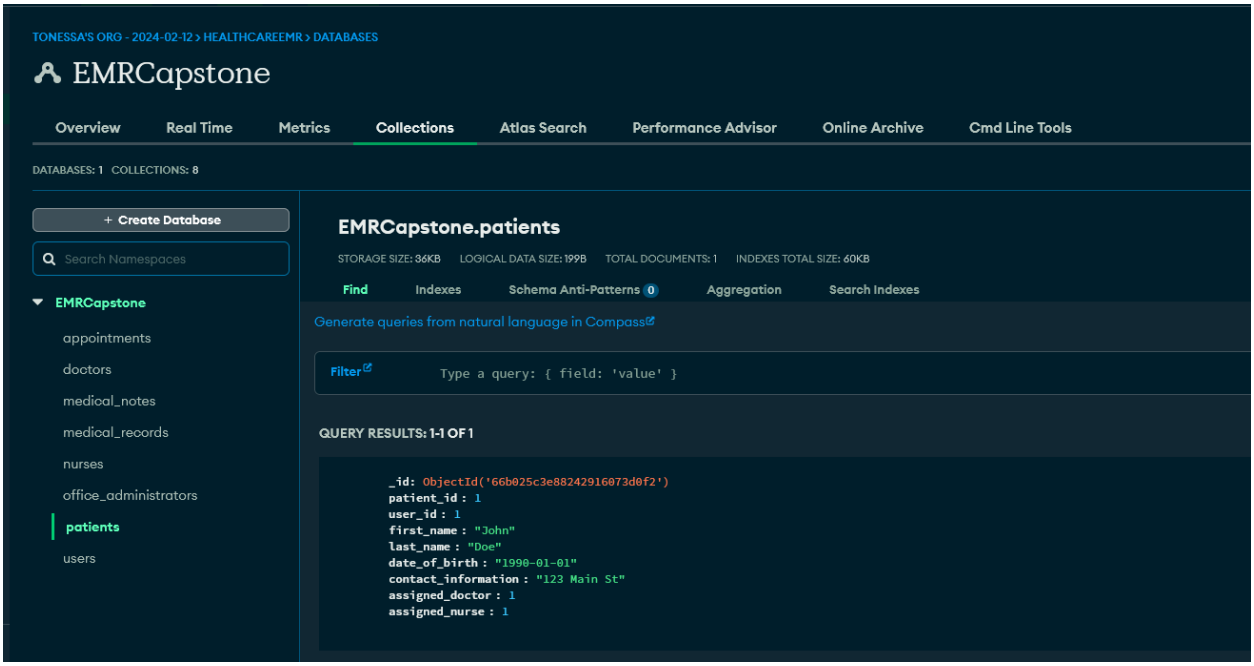
Within the views.py file, I incorporated some proper error handling and as you see below, incorporated more comments within this section. I will be going back to add more comments into other sections before the final. I did receive feedback to move the penalty portion of the code into the .utils for cleaner code organization. I can see that but with the error handling, I'm confused on if that should also move over with it as well?

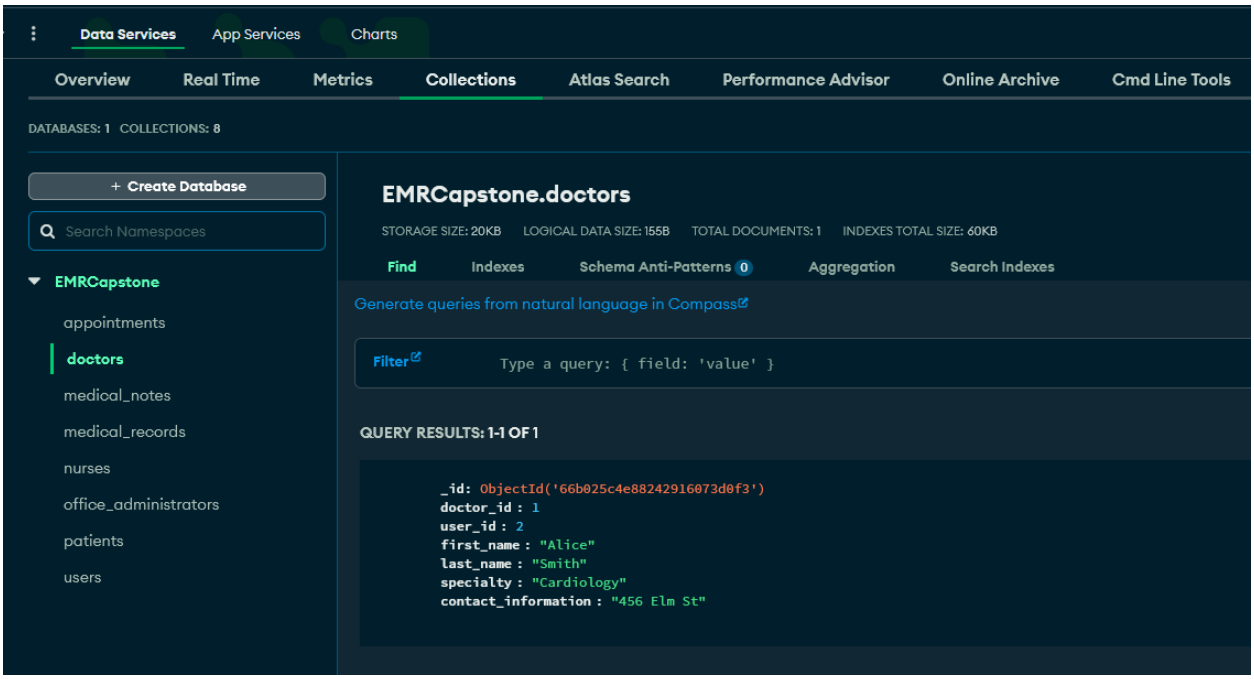
```python
def request_appointment(request):
    if request.method == 'POST':
        try:
            #Retrieves form data
            patient_id = request.POST['patient_id']
            doctor_id = request.POST['doctor_id']
            preferred_datetime_str = request.POST['preferred_datetime']
            reason = request.POST['reason']

            #Parsing out the preferred datetime
            preferred_datetime = datetime.strptime(preferred_datetime_str, '%Y-%m-%d %H:%M')

            #Patient and Doctor Objects
            patient = Patient.objects.get(patient_id=patient_id)
            doctor = Doctor.objects.get(doctor_id=doctor_id)

            #Get available slots for inputted data
            available_slots = doctor.get_available_slots(preferred_datetime.date())

            #If no available slots, return an error
            if not available_slots:
                return JsonResponse({'status': 'error', 'message': 'No available slots'})

            #Calculate the best slot based on penalty
            best_slot = min(available_slots, key=lambda slot: calculate_penalty(preferred_datetime, slot))
            penalty = calculate_penalty(preferred_datetime, best_slot)


            #If the penalty is too high, return an error
            if penalty > 3600:  # 1 hour penalty threshold
                return JsonResponse({'status': 'error', 'message': 'No suitable slots within acceptable range'})

            #Create and book the appointment
            appointment = Appointment.objects.create(
```

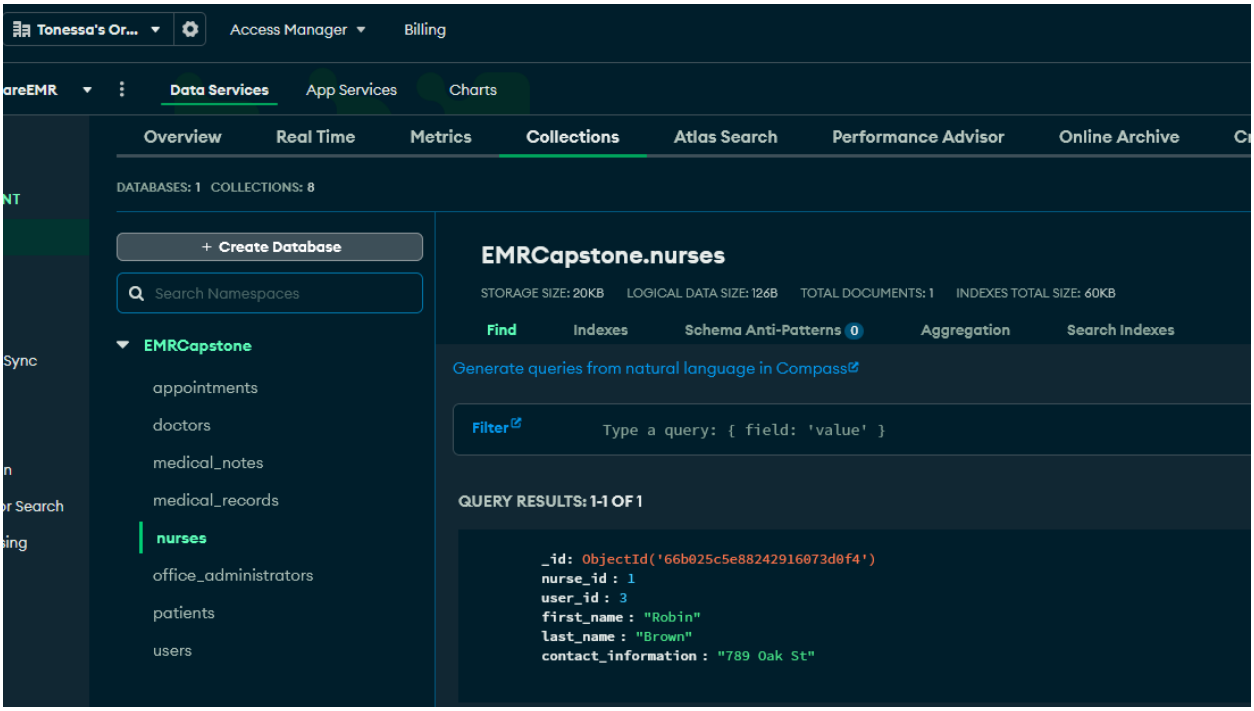
```python
    #Create and book the appointment
    appointment = Appointment.objects.create(
        patient=patient,
        doctor=doctor,
        appointment_date=best_slot,
        reason=reason,
        is_booked=True
    )

    return JsonResponse({'status': 'success', 'message': 'Appointment booked', 'appointment_id': appointment.appointment_id})
```

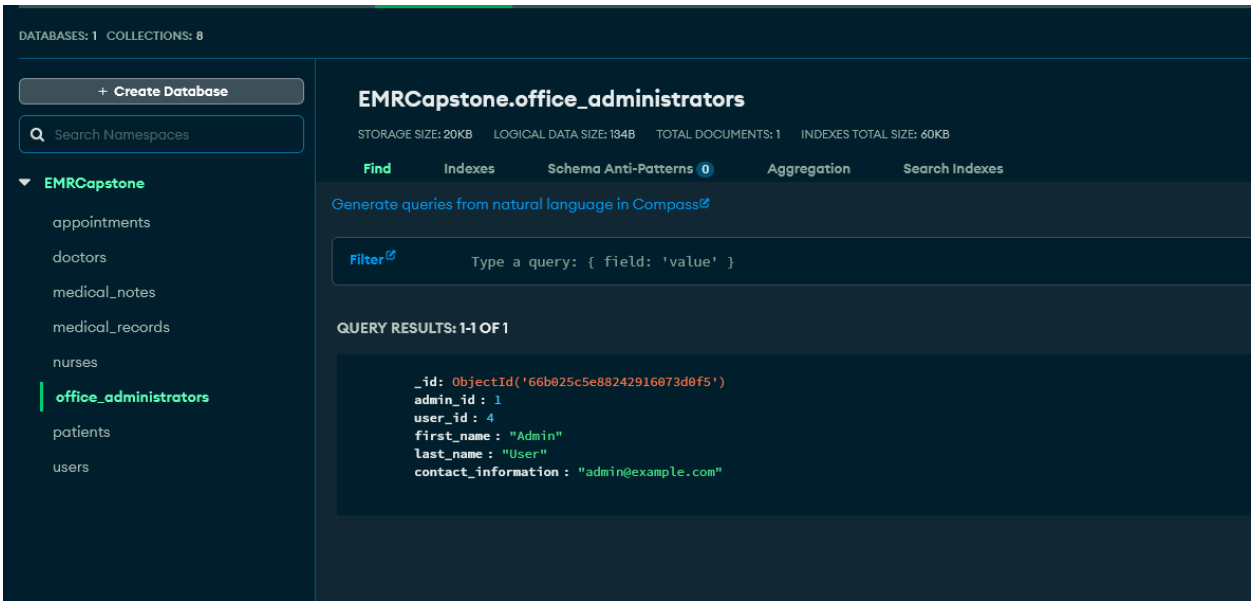
```python
            return JsonResponse({'status': 'success', 'message': 'Appointment booked', 'appointment_id': appointment.appointment_id})

        except ObjectDoesNotExist as oer:
            # Handle case where the patient or doctor does not exist
            return JsonResponse({'status': 'error', 'message': 'Patient or Doctor does not exist: ' + str(oer)})
        except ValidationError as ver:
            # Handle validation errors
            return JsonResponse({'status': 'error', 'message': 'Validation error: ' + str(ver)})
        except Exception as exr:
            # Handle any other unexpected errors
            return JsonResponse({'status': 'error', 'message': 'An unexpected error occurred: ' + str(exr)})

    # Retrieve patients and doctors for the form
    patients = Patient.objects.all()
    doctors = Doctor.objects.all()
    return render(request, 'request_appointment.html', {'patients': patients, 'doctors': doctors})
```

As far as the Database goes, I have the seven collections (essentially, I delved this out to what would need primary key type IDs) and have some records in for the patient, nurse, doctor, and even office administrator. I did the administrator different to show some security issues within the database currently:



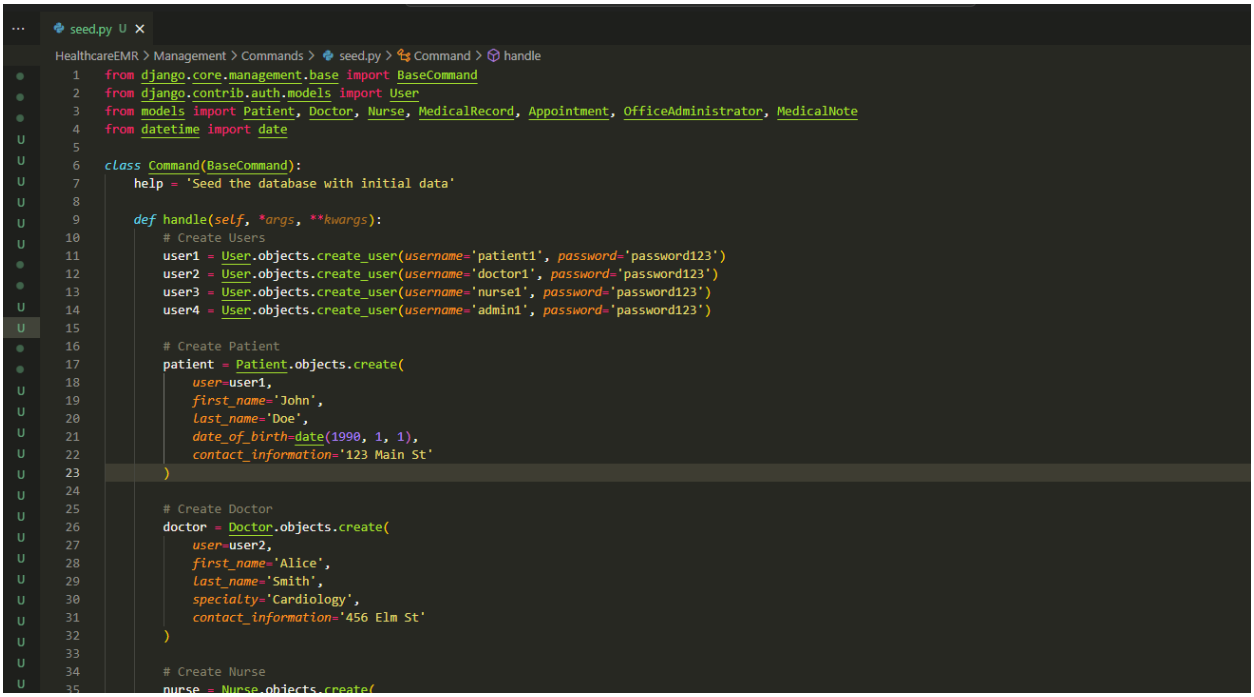






As you can see above, the main issue is that “contact_information” is generic. Are we using address or are we going by email address? If there’s not a specific parameter for the field.

I also incorporated another file called **seed.py**. This seeds the information into the database from the terminal line:


seed.py U
HealthcareEMR > Management > Commands > seed.py > Command > handle
from django.core.management.base import BaseCommand
from django.contrib.auth.models import User
from models import Patient, Doctor, Nurse, MedicalRecord, Appointment, OfficeAdministrator, MedicalNote
from datetime import date
class Command(BaseCommand):
help = 'Seed the database with initial data'
def handle(self, *args, **kwargs):
# Create Users
user1 = User.objects.create_user(username='patient1', password='password123')
user2 = User.objects.create_user(username='doctor1', password='password123')
user3 = User.objects.create_user(username='nurse1', password='password123')
user4 = User.objects.create_user(username='admin1', password='password123')
# Create Patient
patient = Patient.objects.create(
user=user1,
first_name='John',
last_name='Doe',
date_of_birth=date(1990, 1, 1),
contact_information='123 Main St'
)
# Create Doctor
doctor = Doctor.objects.create(
user=user2,
first_name='Alice',
last_name='Smith',
specialty='Cardiology',
contact_information='456 Elm St'
)
Create Nurse

```python
# Create Nurse
nurse = Nurse.objects.create(
    user=user3,
    first_name='Robin',
    last_name='Brown',
    contact_information='789 Oak St'
)

# Assign Doctor and Nurse to Patient
patient.assigned_doctor = doctor
patient.assigned_nurse = nurse
patient.save()

# Create Medical Record
medical_record = MedicalRecord.objects.create(
    patient=patient,
    doctor=doctor,
    nurse=nurse,
    diagnosis='Hypertension',
    treatment='Medication',
    date_of_record=date(2023, 1, 1)
)

# Create Medical Note
medical_note = MedicalNote.objects.create(
    medical_record=medical_record,
    author=doctor.user,
    note_content='Patient needs to follow up in 6 months.'
)

# Create Appointment
```

```python
class Command(BaseCommand):
    def handle(self, *args, **kwargs):
        # Create Appointment
        appointment = Appointment.objects.create(
            patient=patient,
            doctor=doctor,
            appointment_date=None,
            reason='Routine checkup',
            is_booked=False
        )

        # Create Office Administrator
        admin = OfficeAdministrator.objects.create(
            user=user4,
            first_name='Office',
            last_name='Admin',
            contact_information='admin@example.com'
        )

        self.stdout.write(self.style.SUCCESS('Database seeded successfully'))
```

And I also created **init_db.py** to initialize the database an create indexes within the collections (the lavender portion blocks out my actual password to the database):